

Handling Conflicts in Depth-First Search for LTL Tableau to Debug Compliance Based Languages *

Francois Hantry

Mohand-Said Hacid

Université Claude Bernard Lyon 1
LIRIS CNRS UMR 5205
Lyon, France

francois.hantry@liris.cnrs.fr

mohand-said.hacid@liris.cnrs.fr

Providing adequate tools to tackle the problem of inconsistent compliance rules is a critical research topic. This problem is of paramount importance to achieve automatic support for early declarative design and to support evolution of rules in contract-based or service-based systems. In this paper we investigate the problem of extracting temporal unsatisfiable cores in order to detect the inconsistent part of a specification. We extend conflict-driven SAT-solver to provide a new conflict-driven depth-first-search solver for temporal logic. We use this solver to compute LTL unsatisfiable cores without re-exploring the history of the solver.

1 Introduction

Providing adequate tools to tackle the problem of inconsistent compliance rules is a critical research topic. However, few tools fully analyze conflicts over underpinning logics of a natural language (eg. temporal logic, deontic logic...). Such early and declarative specifications can be critical for specifying policies and requirements in agile and distributed environments. Thus, formal languages for compliance requirements and their analysis have become critical in many computer science domains (eg business process management, service oriented computing, e-commerce). An ongoing research topic is the analysis of a conflicting set of temporal logic compliance rules. For instance, Table 1.a gives a toy set of compliance rules. It will be used as a running example in the paper. All those rules except the last one originate from an ongoing supply contract. Let us assume that the last one (r3.c) originates from another internal requirement from a supplier. It comes out that this new requirement entails a conflict with rules (r3.a) and rules (r3.b) shown on Table 1.b. This example shows the importance of automatic detection of conflicting subsets of compliance rules. This problem is critical for debugging declarative specifications [16, 21], handling conflicting contracts [10], or tackling unrealizable service compositions [20]. There exist several formalisms to deal with time such as LTL, MSO [9], TLTL, MTL [1]. These logics underpin many of modern compliance languages and their associated theories and tools are used to address problems related to verification [15, 26, 7], service composition [20], graphical design of property patterns [21, 16].

We investigate the problem of efficiently extracting temporal logic unsatisfiable cores for debugging compliance rules. Intuitively, an unsatisfiable core is a conflicting subset of rules. We restrict ourselves to LTL for which many results and efficient model checking methods exist. However, the problem of efficiently detecting a small LTL unsatisfiable core is still open [23] [6]. Conflict driven methods exist for SAT-solver algorithms. They provide quite efficient extraction of conflicting rules written in propositional logic [28]. SAT-solvers have been extended (e.g., Unbounded Model Checking (UMC) SAT-solvers

*The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

[2]) to deal with the more expressive LTL. For the case of satisfiability¹, it consists in searching a lasso-shaped model of length $k \leq 2^{\mathcal{O}(|f|)}$ and in reducing to boolean SAT problems for increasing $k \in [0, 2^{\mathcal{O}(|f|)}]$. One of critical (and basic) points of current boolean SAT-solvers is their ability of pruning ‘bad’ search space. It is based on a smart use of boolean propagation. Analyzing the propagation enables to handle conflict while backtracking and enables to avoid revisiting immediately the same conflict. Learning conflict using conflict clause also avoids revisiting the conflict later. This conflict-driven approach leads easily to the extraction of a core. [6] proposes to extract unsatisfiable cores from the UMC method of [17]. The authors propose also a ‘Sat Modulo Theory’ like framework applied with symbolic global model checking [5], but the conflict handling is not introduced inside the symbolic global model checking. [23] analyzes a very expanded² tableau of [14] to define unsatisfiable core but again no analysis of conflict is performed. Thus, on the contrary to boolean SAT-solver and extended UMC, neither global model checking, nor On-The-Fly techniques handle conflict. Moreover, in the nineties, resolution [12] for temporal logic has been proposed to tackle unfair SCC as minimal ‘temporal conflict’ but to the best of our knowledge current boolean SAT-Solvers (e.g., [17], [13], [24], [18]) have not investigated this idea yet, mainly because they are Breadth-First-Search. But, a drawback of resolution is that any conflict is recorded using resolvent, this entails a too large use of memory space in contrast to On-The-Fly tableau, symbolic model checking and UMC. In this paper, we propose a new conflict-driven depth-first-search solver inspired by SAT-based ones, DFS for tableau and resolution for temporal logic. Furthermore, we show how it is possible to extract a small unsatisfiable core.

Overview of the paper Section 2 introduces Background. Section 3 describes sound technical details of section 4. Section 4 shows the Solver. Section 5 is devoted to the correctness, completeness, extraction of unsatisfiable cores. We conclude in Section 6.

2 Background

Definition 1 (Syntax of LTL)

Let P be a non empty finite set of propositional variables, and $p \in P$. A and B two LTL formulas. A temporal logic formula is inductively built by means of the following rules:

$$\begin{array}{c} \text{TRUE} \mid \text{FALSE} \mid p \mid A \wedge B \mid A \vee B \mid \neg A \mid X(A) \\ A \cup B \mid A \text{WB} \end{array}$$

Furthermore, $G(A) = (A)W(\text{FALSE})$ and $F(A) = (\text{TRUE})U(A)$.

Definition 2 (Semantic [9]) A linear time structure is an element \mathcal{M} in $(2^P)^{\mathbb{N}}$. $\forall i \in \mathbb{N}, \forall \mathcal{M} \in (2^P)^{\mathbb{N}}$:

- $(\mathcal{M}, i) \models p$ with $p \in P$ iff $p \in \mathcal{M}(i)$
- if A is a propositional combination of LTL formulas $(\mathcal{M}, i) \models A$ is defined as usual.
- $(\mathcal{M}, i) \models X(A)$ iff $(\mathcal{M}, i+1) \models A$
- $(\mathcal{M}, i) \models A \cup B$ iff $\exists j \geq i, (\mathcal{M}, j) \models B$ and $\forall k, i \leq k < j, (\mathcal{M}, k) \models A$
- $(\mathcal{M}, i) \models A \text{WB}$ iff $\forall j \geq i, (\mathcal{M}, j) \models A$ or $(\exists j \geq i, (\mathcal{M}, j) \models B \text{ and } \forall k, i \leq k < j, (\mathcal{M}, k) \models A)$

¹No model to check against a LTL formula

²The expansion disregards boolean conflict

Rules	LTL	Conflicting Rules	LTL core
r1.a	$F(o)$		$TRUE$
r1.b	$G(\neg c)$		$TRUE$
r2.a	$G(o \Rightarrow (F(p) \wedge F(g)))$		$TRUE$
r2.b	$(\neg g)Wp$		$TRUE$
r3.a	$F(i)$	r3.a	$F(i)$
r3.b	$(\neg i)Wp$	r3.b	$(\neg i)Wp$
r3.c	$G(p \Rightarrow G(\neg i))$	r3.c	$G(p \Rightarrow G(\neg i))$

(a) *compliance rules.* (b) *unsatisfiable core.*

Table 1: LTL-translations of the running example

Intuitively, the formula $X(A)$ stands for ‘at the next time A will hold’, AUB stands for ‘ B will hold in the future and from current time until B holds, A must hold’, AWB stands for ‘if B holds in the future then from current time until B holds, A must hold, and if B will never hold, then A must hold forever (weak until)’. $G(A)$ stands for ‘at any time A holds’ and $F(A)$ stands for ‘ A will hold in the future’. For instance $\neg iWp$ means that i cannot occur as long as p has not occurred.

In the rest of the paper we will assume w.l.g that any LTL formula solely may contain \neg symbol applied to propositional variable(s). We call such formula Negative Normal Form (NNF).

Definition 3 (LTL SAT problem) A LTL formula ϕ is satisfiable iff there exists a linear model M such that $(M, 0) \models \phi$. Conversely, a LTL formula ϕ is unsatisfiable iff there is no linear model M such that $(M, 0) \models \phi$.

Definition 4 (unsatisfiable core) An unsatisfiable core of an unsatisfiable formula ϕ is a formula ϕ' such that (1) ϕ' is the result of some substitution(s) in ϕ of some positive subformula(s) by $TRUE$, (2) ϕ' has no subformula of the form $AU/W(TRUE)$, $(TRUE)WB$, $A \vee (TRUE)$, $\wedge_i True$ or $X(TRUE)$ and (3) ϕ' still remains unsatisfiable. Table 1.b shows a small unsatisfiable core of our toy example formula. It is critical to find a small (or ideally a minimal³ (MU)) unsatisfiable core in order to detect the cause of a conflict.

Theorem 1 (LTL minimal unsatisfiable core decision problem)

Deciding if a LTL formula is a minimal unsatisfiable core is in P-SPACE

(**sketch of the proof**): For each positive subformula of f , substitute by $TRUE$ and check unsatisfiability. f is a MU iff any substitution leads to a satisfiable formula. There is a linear number of subformulas, and each checking is in P-SPACE.

We furthermore conjecture that the above problem is P-SPACE complete.

[23] discusses the notion of granularity of core. A coarse unsatisfiable core of a formula $f : f_1 \wedge f_2 \dots \wedge f_n$ only substitutes $TRUE$ at the f_j and not in a deeper subformula. Structure preserving translations of the LTL formula f into definitional conjunctive normal form provide an *equi-satisfiable* formula $f' : f'_1 \wedge f'_2 \dots \wedge f'_m$. The minimal coarse unsatisfiable cores of f' correspond to the minimal unsatisfiable cores of f (see [23] for details). For instance if $f : G(a \wedge \neg b) \wedge F(b)$, an equi-satisfiable formula may be $f' : G(x_{a \wedge \neg b}) \wedge G(x_{a \wedge \neg b} \Rightarrow a) \wedge G(x_{a \wedge \neg b} \Rightarrow \neg b) \wedge F(b)$. A coarse MU of f' is $G(x_{a \wedge \neg b}) \wedge TRUE \wedge G(x_{a \wedge \neg b} \Rightarrow \neg b) \wedge F(b)$. It provides a f -MU : $G(TRUE \wedge \neg b) \wedge F(b)$. W.l.g [23], the solver will focus

³An unsatisfiable core ϕ is minimal iff ϕ is its only one unsatisfiable core

on finding small coarse unsatisfiable core.

Definition 5 (Closure) Let f a LTL formula. We note the set of closure variables of f - $Cl(f)$ - as the smallest set Set such that :

- $f \in Set$
- If $\psi = \psi_1 \wedge .. \wedge \psi_s \in Set$ and ψ_j is not a conjunction, $\forall j \psi_j \in Set$
- If $\psi = \psi_1 \vee .. \vee \psi_r \in Set$ and ψ_j is not a disjunction, $\forall j \psi_j \in Set$
- If $\psi = F/G(\psi') \in Set$, $\psi' \in Set$ and $XF/G(\psi') \in Set$
- If $\psi = \psi'U/W\psi'' \in Set$, ψ'' and $\psi' \wedge X(\psi)$ are in Set
- If $\psi = X(\psi') \in Set$ then $\psi' \in Set$

Furthermore the number of closure variables of $Cl(f)$ is linear in the size of f [11].

A traditional mathematical tool to analyze satisfiability is tableau. It is a particular automata of states, whose any state is a subset of $Cl(f)$. Intuitively, a state is built from a prestate. A prestate is either the starting state containing only the starting formula f either a state containing only closure formulas derived from a precedent state. The derivation of a formula Xh at a state is h at the next prestate. The prestates are intermediary results to build the tableau and do not occur in the tableau except the first one. On Figure 1, the rounded rectangle is a prestate, the others are states. A state is computed by unwinding the formulas and making a choice for the disjunctive ones. For instance, the occurrence of $G(\neg i)$ implies the occurrence of $\neg i$ and $XG(\neg i)$. In Figure 1, at the goal state of transition 1, the p is chosen from the disjunction $p \vee (\neg i \wedge X(\neg iWp))$ unwound⁴ from $\neg iWp$.

Definition 6 (state, prestate, f -tableau) The f -tableau is a special finite state automata (St, s_0, R) with St the set of states, s_0 the initial state and $R \subset ST^2$ the set of transitions. The f -tableau is the 'minimal' automata A such that:

- Any state of A is a subset of $Cl(f)$.
- s_0 is a prestate with $s_0 = \cup_i \{f_i\}$ where $f = \wedge_i f_i$.
- Let a set S derived from a prestate PS st. $PS \subset S \subset Cl(f)$ and $\exists \rho$ a total choice function from $S \cap (disjunction \cup Future \cup Until \cup WUntil)$ to S . Furthermore, if S is the smallest set Set such that
 - $PS \subseteq Set$
 - If $\psi = \psi_1 \wedge .. \wedge \psi_s$ and ψ_j is not a conjunction, $\forall j \psi_j \in Set$
 - If $\psi = \psi_1 \vee .. \vee \psi_r$ and ψ_j is not a disjunction, $\rho(\psi) = \psi_j \in Set$ for some j
 - If $\psi = F(\psi')$, $\rho(\psi) \in \{\psi'; XF(\psi')\} \cap Set$
 - If $\psi = G(\psi')$, $\psi' \in Set$ and $XG(\psi') \in Set$
 - If $\psi = \psi'U/W\psi''$, $\rho(\psi) \in \{\psi''; \psi' \wedge X(\psi)\} \cap Set$,

then S is a state of A .

- Let a set PS containing only all formulas derived from a precedent state S such that $PS = \{\phi, st.X\phi \in S\}$. Then, PS is a prestate of A .
- Transitions R of a f -tableau stand for the collapsing of $S_1 \rightarrow PS \rightarrow S_2$ derivation sequences, i.e., collapsed transitions of the form $S_1 \rightarrow S_2$.

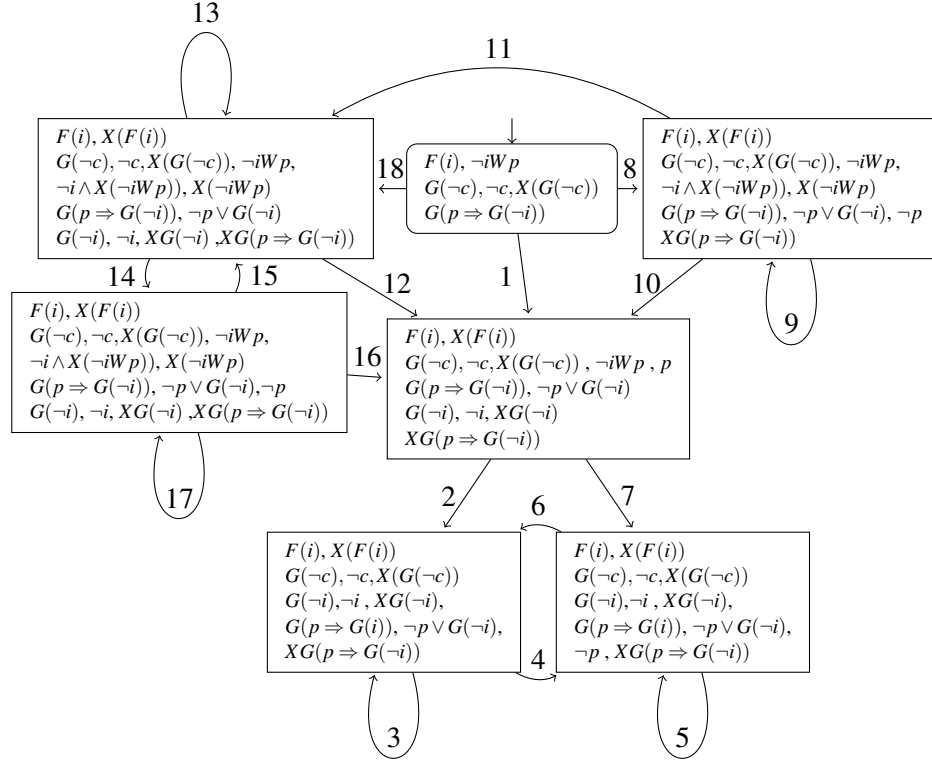


Figure 1: Depth-first-search

Theorem 2 ([19],[14]) A LTL formula f is satisfied iff there exists a path of states in the f -tableau (finite with no successor at the last state or infinite) starting from the starting prestate and such that any occurrence of Future and Until modal operator in a state of the path fulfills its corresponding promise operand later (in the future) in the path. We call the path : fair path.

In Figure 1, f is a simpler version of our toy example, and there is only unsatisfiable paths (infinite in this case) since each possible path contains a Future $F(i)$ but does not realize the promise operand i . An argument is that any infinite path will reach in the future a Strongly Connected Component (SCC) where the path will remain in forever. Then f is unsatisfiable. On-the-fly techniques for satisfiability of temporal logic (eg. [14], [19]) use nested deep-first-search of fair loop or simple deep first search of fair SCC.

Theorem 3 ([25],[19]) There exists a depth-first-search algorithm for computing SCCs of a f -tableau, and for deciding their fairness.

In Figure 1, the exploration steps of simple depth-first-search follow the numbered labels on the transitions. An example of a SCC is the set of states as a support for the set of transitions $\{3;4;5;6\}$.

⁴disjunctive unwinding are not shown in the tableau since this is an intermediary result

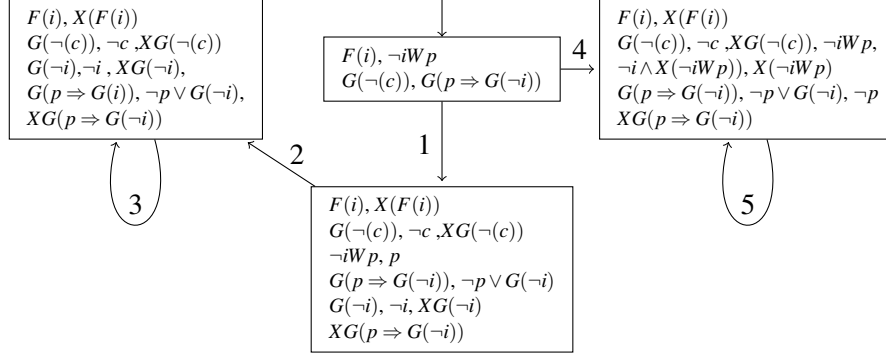


Figure 2: Depth-first-search with conflict handling and prime implicant

3 Technical Preliminaries

We will show how it is possible, by handling conflicts, to enhance above depth-first search method and to drastically shrink the search space. Our solver shown in Section 4 is based on the following intuitions. First, the idea is to record which occurrences of elements of the closure at a given state entail another one by using unit rule propagation technique from SAT-Solvers. It enables to extract cause of conflict and non-chronologically backtrack at the last involved choice and eventually to learn information from the conflict, in order to not revisit the same conflict. Furthermore, our solver uses fair prime implicant search to also shrink good but redundant search space. These optimizations enable to only explore the tableau of Figure 2 to decide unsatisfiability of the running example f – *tableau* of figure 1. In the following we explain how unwinding from prestate to state is simulated by a boolean SAT-problem.

3.1 From prestate to state: a propositional SAT problem

To tackle the particular choice function handling at Definition 5 (one literal per occurring disjunction) we need a ‘three-values’ logic which enables partial instantiation. It is also convenient for prime implicant handling.

Definition 7 (‘three-value’ logic, closure variables, literals, clause) Let S be a set of LTL formulas. We call state closure of S $StCl(S)$ any formula met in Set of the closure algorithm with the initial condition on $Set = S$ instead of $Set = \{f\}$ and without the last rules (Xg derives g). For any element g in the closure we note x_g a fresh boolean variable, that we call closure variable. This means presence of g in the state. We will use the word ‘literal’ for x_g or $\neg x_g$. Finally, we call a clause a disjunction of such literals (also represented by a set of literals). Let $S' \subset Cl(f)$. We say that S' is conflicting if there exists h and $\neg h$ in S' . Let V be a set of closure variables, L the literals of V . Then if g and h are ‘three-values’ logic formulas then $x_{h'} \in V$, $g \wedge h$, and $\neg g$ are ‘three-values’ formulas. Furthermore assuming S' is non-conflicting :

- $S' \models x_{h'}$ iff $h' \in S'$
- $S' \models g \wedge h$ iff $S' \models g$ and $S' \models h$
- $S' \models \neg g$ iff $S' \not\models g$

We say that a three-values formula g is valid iff for any non-conflicting set of S' , $S' \models g$. We say that g is fair-valid if for any S' which is a state from any fair path $S' \models g$.

Definition 8 (Unwinding clauses from a prestate) Let PS a prestate and $Presence(PS) = \{x_h | h \in PS\}$. The corresponding Unwound Clause Set $UCS(PS)$ is a set containing the unwound clauses and $AUX(PS)$

the three values conditions. Set , $AUX(PS)$ and $UCS(PS)$ are the smallest sets following the rules :

- $Presence(PS) \subseteq Set \cap UCS(PS)$
- If $x_\psi = x_{\psi_1 \wedge \dots \wedge \psi_s} \in Set$ and any x_{ψ_j} is not a conjunction,
 $\forall j$ the formulas $x_\psi \Rightarrow x_{\psi_j} \in UCS$ and $\forall j x_{\psi_j} \in Set$
- If $x_\psi = x_{\psi_1 \vee \dots \vee \psi_r} \in Set$ and any x_{ψ_j} is not a disjunction,
 $x_\psi \Rightarrow (x_{\psi_1} \vee \dots \vee x_{\psi_r}) \in UCS$ and $\forall j x_{\psi_j} \in Set$
- If $x_\psi = x_{\psi'} U / W x_{\psi''}$,
 $x_\psi \Rightarrow (x_{\psi'} \vee (x_{\psi'} \wedge x_{\psi''})) \in UCS$ and $x_{\psi''}$ and $x_{\psi' \wedge x_{\psi''}} \in Set$
- $x_h, x_{\neg h} \in Set$ then $\neg x_h \vee \neg x_{\neg h} \in AUX$

Furthermore, $AUX(f)$ (resp. $UCS(f)$, $Presence(f)$) is the union of $AUX(PS)$ for any PS in the f -tableau (resp $UCS(PS)$, $Presence(PS)$). The unwound formulas $UCS(PS) \setminus Presence(PS)$ are fair valid formulas (see proof section 5) and of the form $x_\phi \Rightarrow disj_{x_\phi}$ where $disj_{x_\phi}$ is the classical disjunctive unwinding of closure formulas [14], [19].

The formula f of Figure 3 provides the clause $UCS(f)$:

$$\begin{array}{l|l}
 x_{F(i)} \Rightarrow x_i \vee x_{XF(i)} & x_{G \neg c} \Rightarrow x_{XG \neg c} \\
 x_{G \neg c} \Rightarrow x_{\neg c} & x_{(\neg i)Wp} \Rightarrow x_p \vee x_{\neg i \wedge X((\neg i)Wp)} \\
 x_{\neg i \wedge X((\neg i)Wp)} \Rightarrow x_{\neg i} & x_{\neg i \wedge X((\neg i)Wp)} \Rightarrow x_{X((\neg i)Wp)} \\
 x_{G(p \Rightarrow G(\neg i))} \Rightarrow x_{XG(p \Rightarrow G(\neg i))} & x_{G(p \Rightarrow G(\neg i))} \Rightarrow x_{p \Rightarrow G(\neg i)} \\
 x_{p \Rightarrow G(\neg i)} \Rightarrow x_{\neg p} \vee x_{G(\neg i)} & x_{G(\neg i)} \Rightarrow x_{XG(\neg i)} \\
 x_{G(\neg i)} \Rightarrow x_{\neg i} & \forall v \in CLST f \neg x_v \vee \neg x_{\neg v}
 \end{array}$$

Proposition 1 An instance IS of the boolean SAT problem $UCS(PS) \cup AUX(PS)$ provides a state S from PS and reciprocally.

Since many instances correspond to a state in the tableau, and since several states may be redundant regarding LTL satisfiability problem, we introduce Fair Prime Implicant.

Definition 9 (Fair Prime Implicant) Let IS as above, a Fair Prime Implicant $IS.FPI$ of IS is a maximal⁵ switching from some assigned x_h at IS to $\neg x_h$ such that h is not a promise operand and $IS.FPI \models UCS(PS) \cup AUX(PS)$. At a given $IS.FPI$ it corresponds only one state FPI in the f -tableau.

Theorem 4 (Fair prime implicant version of Depth-First-Search) A formula f in LTL is satisfiable iff there exists a fair path solely with FPIs as states. (*proof is omitted*).

For instance, the FPI technique enables in our depth first search to ignore the goal state of the transition number 4 at Figure 1.

To solve the boolean SAT-problem current solvers use unit rule propagation [8].

Definition 10 (Unit rule propagation)

- Each instantiated literal must be propagated⁶ over any non yet satisfied clause containing the opposite one. This opposite literal is then temporally erased from the clause.
- If a clause becomes unit literal l because of unit rule propagation(s), then l is assigned

This propagation is critical for conflict analysis. In the following we show how to handle unit rule propagations to support conflict analyses.

⁵Intuitively the switching simulates the removal of closure element in corresponding state

⁶a Weakest version and optimized one of current solvers requires only propagation along watched literals [22]

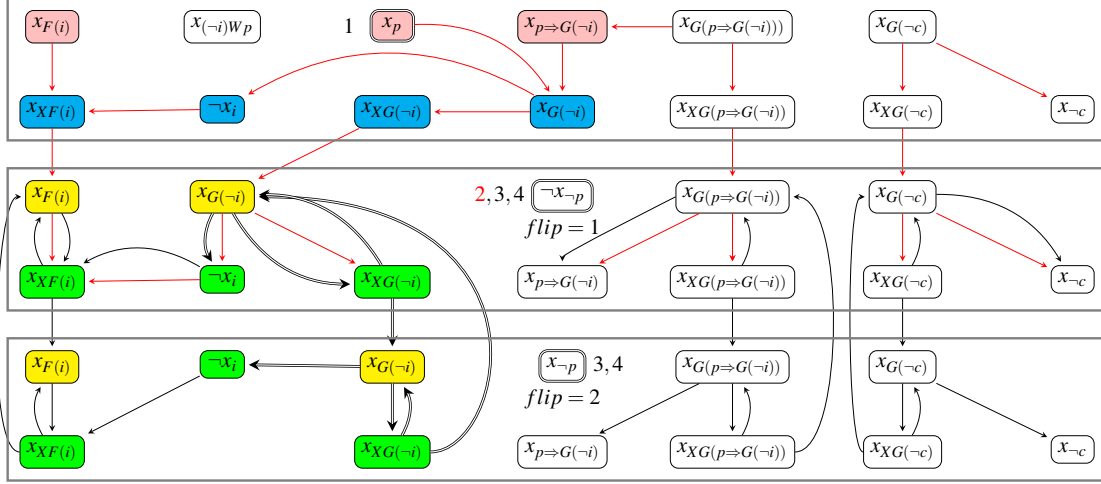


Figure 3: Implication graph and conflict analyses

3.2 Implication Graph to support Conflict analyses

The Implication Graph is an extension of propositional SAT-solvers' one to LTL-tableau. The intuition is to record the occurrences of elements of the closure at a given state that entail another one. An Implication Graph is a bicolor graph $(Nodes, T_{red}, T_{black})$ where T_{red} and T_{black} are subsets of $Nodes^2$. Figure 3 shows a part of the Implication Graph adapted from the f -tableau of Figure 1. Intuitively, the Implication Graph is a concatenation of several IS s implication graphs denoted $IS.IG$. The red part T_{red} is used for conflict analysis of the depth-first-search stack S and it is a DAG, and the black part T_{black} records some past red edges and corresponds to the conflict analysis of the SCC-search using stack S' and allows loop for inductive reasoning .

Nodes' feature Intuitively, a Node N stands for an assigned literal at a given state. On Figure 3, the rounded corners rectangles are Nodes. Each node is inside a big rectangle standing for state. More precisely, a Node corresponds to the ongoing prestate, to an ongoing IS while it is found and to a chosen extracted $IS.FPI$ in this case. On Figure 3, the three states⁷ are the one which support the transitions $\{2; 3; 4\}$ on Figure 1. Furthermore a Node can be either chosen or required. On Figure 3, a chosen node is doubly surrounded. The level of a chosen Node N is its chronological order of choice in the whole f -tableau. On figure 3 numbers are levels of chosen nodes. The level of any node N is the maximum⁸ level of the choosen nodes which involve N i.e which are ancestors of N in T_{red} . The level of a set of nodes is the maximum level its nodes. A required node is either without antecedent but with level 0 either gets an antecedent in T_{red} .

Transitions' feature If a Node N_l which corresponding literal l comes from a clause $C = \bigvee_j l_j \vee l$ which has become unit, then the red edges $(N_{\sim l_j}, N_l)$ are in T_{red} just after this unit propagation. Let's focus on the above state. For instance, $(x_p, x_{G(\neg i)})$ and $(x_{p \Rightarrow G(\neg i)}, x_{G(\neg i)})$ are red edges because of the unit rule from the clause $x_{p \Rightarrow G(\neg i)} \Rightarrow (x_{\neg p} \vee x_{G(\neg i)})$. Furthermore, the derivation from a state to a next state is also recorded using red edges such that the occurrence of $x_{X_g} \in IS.FPI$ entails the occurrence of x_g at the next prestate. For instance on Figure 3, the above FPI derives to the middle one, thus there exists a red edge in the graph from $x_{X_F(i)}$ at above state to $x_{F(i)}$ at the next one.

⁷for understanding but w.l.g, the below state is not a FPI

⁸0 if no ancestor nodes are choosen

Furthermore, while a FPI is revisited, then the current *IS* implication graph *IS.IG* has to be connected to the first one *IG_{old}* which visited the same FPI. The algorithm creates black transitions from any nodes $N(\neg x_{op_{pro}})$ (resp. $x_{Xh} \in IS.FPI$) at *IS.IG* to the same literal one of *IG_{old}*. This connection is called ‘bind’ function. For instance, for corresponding derivation on Figure 1 for *IS.IG* at the goal state of transitions $\{3;5;6\}$ the *IS.IG* and *IG_{old}* are the same. For simplicity and w.l.g.⁹ they have been superimposed at Figure 3. In this case, the transitions of ‘bind’ have been omitted w.l.g. and solely the bottom-up edges from source state to the goal state of transitions $\{3;5;6\}$ are shown (e.g., $x_{XF(i)}$ at the below state to $x_{F(i)}$ at the same state for transition 3). Finally, given a T_{red} and a choosen Node N still red, $flip(Nodes, T_{red}, T_{black}, N) = (Nodes \cup \{\sim N(red)\}, T_{red} \setminus \{(N_1, N_2) \in T_{red} \mid level(N_2) \geq level(N)\}, T_{black})$ is the flipped Implication Graph regarding N with $\sim N(red)$ a fresh node.

4 Solver

Our depth-first search temporal conflict driven solver is a combination of depth first search of fair SCC in tableau [19] and of boolean SAT-solver. Thus, our solver uses unit rule propagation method, boolean conflict handling [22]. It also uses a new temporal conflict driven method inspired by resolution for temporal logic [12].

Basic Solver¹⁰ Algorithm 1 shows the main method of the algorithm called Solver. At each new prestate, the solver populates by clauses by unwinding the prestate according to Definition 8. Otherwise, unit rules and boolean conflict detection¹¹ are launched. A Backtrack (Algorithm 2) is triggered in case of a conflict, otherwise if it is possible, a choice of literal following a heuristic is done. Once an *IS* is found and a FPI extracted, then a SCC-search-forward (Algorithm 3) function is called. Otherwise the Solver is recursively called.

Algorithm 1: Solver

if not unwound then Unwind; Unit-rule ; bool-conflict-detection ; if conflict then Backtrack;	if IS found then SCC-search-forward; else make a choice of literal; Solver ;
---	--

Propositional Conflict Handling while backtracking A Propositional Conflict Handling is triggered when a clause is falsified (or equivalently when a literal and its opposite occurs). Similarly to SAT-solvers’ one, the Propositional Conflict Handling starts from a set of conflicting nodes $Nodes_C$ and corresponding literals C which falsifies the clause $\neg C$ and analyzes which nodes have involved those conflicting literals using $(Nodes(red), T_{red})$. Let $\mathcal{A}(C)$ be the subDAG of $(Nodes(red), T_{red})$ which stands for ancestors of $Nodes_C$. Let $\mathcal{A}(C)(conflict - level)$ be the subDAG of $\mathcal{A}(C)$ with nodes of ‘conflicting’ level of $Nodes_C$ ie. $conflict - level$ and $N(conflict - level)$ the choosen node of level $conflict - level$. Let $Limit(C) = \{N(conflict - level)\} \cup (Parent_{T_{red}}[\mathcal{A}(C)(conflict - level) \setminus \{N(conflict - level)\}] \cap Level(conflict - level - 1, \mathcal{A}(C)))$ where $Level(m, \mathcal{A}(C))$ means the subDAG of $\mathcal{A}(C)$ with node level

⁹The particular computation of fixpoint remains the same while superimposing in this simple case

¹⁰The main components of the algorithm are shown in a recursive form for convenience

¹¹ a boolean conflict detection occurs while a clause is falsified by current partial assignment

at most m . We call limit conflict clause $\neg Limit(C)$. The last conflicting chosen node $N(conflict - level)$ is then switched if the corresponding flipped partial assignment has not been visited yet (node.flip=1). In this first case, similarly to boolean SAT-solvers, the function ‘Conflict-require’ adds red edges to $(Nodes(red), T_{red})$: the red transitions with a source node in $Limit(C) \setminus \{N(conflict - level)\}$ to the goal node $\sim N(conflict - choice)$. However, differently from boolean SAT-solver, since the algorithm records informations in black part $(Nodes(black), T_{black})$ in the second case (flip=2), the same transitions but in color black are added. Furthermore, $\sim N(conflict - level)$ is now required and not chosen. Those red or black edges are to ensure we can compute the reason of the requirement of $\sim N(conflict - level)$. Finally, if the conflict level is 0 then the algorithm terminates by unsatisfiable.

Algorithm 2: Backtrack

Compute Conflict-level;	stack-s.erase(C-level);
if $Conflict-level=0$ then	stack-s'.erase(C-level);
print (‘unsat’) , break;	Conflict-require;
State-Conflict-Clause-learning;	SCC-search-backward;
Tableau.IG.erase(Conflict-level);	

On Figure 3, the backtrack is done from the conflicting (see. TC-Analysis) nodes $x_{G(\neg i)}$ and $x_{F(i)}$ at the middle state. Following the red part, the last involved and chosen node is x_p at above state. While backtracking bad states and corresponding nodes are erased (above state at Figure 3). On the contrary to propositional SAT-solver, the algorithm has to record the cause of these states to be bad (to avoid revisiting them) using a conflict clause per state¹². These learned clauses must not be forgotten. On figure 3, the yellow literals are conflicting literals at middle state but the clause $\neg x_{G(\neg i)} \vee \neg x_{F(i)}$ has already been learned. At above state, the pink literals provide the learned clause $\neg x_p \vee \neg x_{p \rightarrow G(\neg i)} \vee \neg x_{f(i)}$. Finally, a SCC-search backward is launched. Algorithm 2 summarizes the above ideas. We refer to [27] for more details about backtracking in boolean SAT-solvers.

Algorithm 3: SCC-search-forward

if FPI is new then Nb(FPI):=i:=i+1; Lp(FPI):=Lv(FPI:=Nb(FPI)); stack-s.push(FPI); stack-s'.push(FPI); parent=FPI; prestate=FPI.next(); Solver;	else case $state \in stack - S$ Lp(Parent):= min(Lp(Parent),Nb(FPI)); case $FPI \notin stack - S \wedge$ $Nb(parent) > Nb(FPI)$ Lv(Parent) :=min(Lv(parent),Nb(FPI)); parent.unr-prom= parent.unr-prom \cap $FPI_{old}.unr-prom$; bind($IS.IG, IG_{old}$); SCC-search-backward ;
--	---

SCC-Search-Forward The SCC-search-forward shown Algorithm 3 is similar to the ‘forward’ part of the computation of strongly connected components and uses depth first search numbers (Lp,Nb,Lv). If the FPI is new, then new numbers are computed and if it is possible, the next prestate (and corresponding

¹²We ask that the conflicting clause forbids corresponding red FPI of state

prestate Nodes and transitions from derivations) are created from the (red) nodes from literals $x_{Xh} \in IS.FPI$, otherwise the problem is satisfiable. Moreover, if the already visited FPI_{old} is still in $Stack - S'$ or in $Stack - S$, a computation on Tarjan's numbers¹³ is also launched. The unrealizable promises are also computed. Furthermore, in any revisiting case, a rollback is launched while calling SCC-search-backward (see Algorithm 4).

SCC-Search-Backward First the algorithm adds black copies of red edges in $IS.IG$. Then, starting with the current chosen node N of current level, the Algorithm 4 simply finds the last non-flipped chosen node. If it is in IS then, it calls $flip(IG, N)$ and Solver. Otherwise change color red to black at the 'next' edges from $parent.IG$ to IG . Then a SCC test over Tarjan numbers is launched from the parent state, and if a SCC is found a SCC-handling is called, otherwise, update of unrealizable promise is done. If a promise is unrealizable then SCC-handling calls a Temporal Conflict Analysis (TC-Analysis), otherwise the problem is satisfiable.

Algorithm 4: SCC-search-backward

```

N=node(level)
IS.IG.edges.black-copies
if  $N.flip=2 \wedge N \in IS$  then
  level=level-1; SCC-Search-Backward
if  $N.flip=1 \wedge N \in IS$  then
  flip(IG,N)
  Solver;

if  $N \notin IS$  then
  red-to-black-parent.IG-IS.IG-derivation
  FPI=parent; pop stack-s
  parent= head stack-s
  if  $Lp(FPI)=Nb(FPI)=Lv(FPI)$  then
    SCC-handling*
  else
    Lp(parent)=min(Lp(parent),Lp(FPI))
    Lv(parent)=min(Lv(parent),Lv(FPI))
    parent.unr-prom= parent.unr-prom  $\cap$ 
    FPI.unr-prom
    SCC-search-backward

SCC-handling* ::
if  $unrealizablepromise = \emptyset$  then
  print 'satisfiable'; break;
else
  TC-Analysis;
  
```

TC-Analysis of unfair SCC In the SCC, the algorithm 5 chooses an unfair promise and computes a backward fixpoint from some nodes $N(\neg x_{op(Promise)})$ for any SCC states along the recorded black implication graph. Precisely, except the root state of the SCC, any state of the SCC gets a corresponding black 'IG' from $stack - S$ which is the $IS.IG.edges.black - copies$ one while SCC-backward-search. For the root state SCC, only the nodes $N(x_{Xh})$ and $N(\neg x_{op(Promise)})$ get some black transitions.

The fixpoint computation starts from those nodes at $IS.IG.edges.black - copies$ or particular nodes at the root. Once the inflationary backward fixpoint using T_{black} is terminated, then at each state in SCC, the algorithm picks up a corresponding¹⁴ IG . For any state, the 'prestate(s)' Nodes $Nodes_{prestate}$ in the IG which are also in the fixpoint are declared conflicting with the unfair promise and the algorithm learns and must not forget the conflict clause. Then, the method erases all the states of this SCC. It finally triggers a classical Backtracking at the nodes of the *Root* from the conflicting prestate(s)Nodes of the

¹³Please see for more details about Tarjan's numbers [25]

¹⁴Since the root has been revisited, it gets at least one black IG

Algorithm 5: Temporal Conflict Analysis

INI: $Vector = \neg ops(Promise) \cap SCC;$ while $\exists e \in Vector \wedge e$ not marked do mark e ; $v = e.black - parents$; for $l \in v \wedge l$ not marked do $Vector.push(l)$ end	$\forall state \in SCC$ pick up a $State.IG$; do learn($Vector \cap state.IG.prestate, promise$); erase SCC ; Backtrack;
---	--

root. At Figure 3, the unfair promise is $F(i)$, and the fixpoint computation is shown by double arrow. In this SCC, the yellow and green Nodes are involved in the temporal conflict, and the yellow are the causes of this conflict, ie., $x_{F(i)}$ and $x_{G(\neg i)}$ are conflicting. Thus, $\neg x_{G(\neg i)} \vee \neg x_{F(i)}$ is learned forever.

5 Correctness, Completeness, and Extraction of a small unsatisfiable core

Lemma 1 Any clause from $AUX(f)$ or $UCS(f) \setminus Presence(f)$ are fair valid.

proof: Any fair state is non conflicting then AUX is fair valid. By construction, any fair state satisfies any clause from $UCS(PS) \setminus Presence(PS)$.

Lemma 2 Let f be a LTL formula. Assume the Algorithm has computed a conflict analysis from the conflicting literals C . Let $ICl(f, C) = AUX(f) \cup UCS(f) \cup Learn(f, C)$ with $Learn(f, C)$ containing any learned clause occurring in the algorithm strictly before C and any limit conflict clause¹⁵ occurring at any conflict handling strictly before C . Assume that $Learn(f, C)$ are fair valid clauses. Let Cf be the conjunction of conflicting literals used to learn a resulting clause of the conflict analysis $\neg Cf$. Then $\neg Cf$ is fair valid.

sketch of the proof: Thanks to lemma 1, $ICl(f, C) \setminus Presence(f)$ are fair valid clauses. Let any state S from any fair path p of any tableau of a temporal logic formula. Assume now that $S \models Cf$. We have two cases:

1. Either the conflict C is boolean. Let $\mathcal{A}'(C) = Level(\mathcal{A}(C), conflict - level)$ and $Limit(C)$ as above. Then each node in $\mathcal{A}'(C) \setminus \{n(conflict - level)\}$ is required and it originates either from state to prestate derivation, either from a clause $Cl \in ICl(f, C) \setminus Presence(f)$ which has become unit at a given state. Since $ICl(f, C) \setminus Presence(f)$ are assumed fair valid then $S' \models Cl$ for such a clause Cl and for any state S' in p_S ¹⁶. Then the proof from $\mathcal{A}'(C)$ by unit rule of the conflict C of our algorithm implies that there exists a state $S'_{conflict}$ in p_S such that $S'_{conflict}$ contains \square . This implies a contradiction since p is assumed fair and then no state of p should be conflicting.
2. Either the conflict C is temporal. Assume S' any state of the unfair SCC. For any state of the SCC, let $Pre(S')$ be the set of 'black' prestates from a chosen $IS.IG$. from S' . Let $k \in \mathbb{N}$. Imagine virtually the exploration of any non conflicting prefix path p' of length k in the induced tableau $T(Pre)$ by also considering $ICl(f, C) \setminus Presence(f)$. It consists of the building of a boolean SAT-problem based on the following observations:
 - Since any bad old SCC is not reachable by not forgetting any conflict clause of bad state/bad SCC, then there exists a k -depth-first navigation over the Prime Implicants from $T(Pre)$ but

¹⁵ the limit conflict clause is $\neg Limit(C)$; we consider it even if the limit conflict clause is not learned by the solver

¹⁶ suffixes of p from S

remaining in the unfair SCC and following the Prime implicant depth-first-search of the f -tableau.

- if $(s_{i_0} = Pre, s_{i_1}, \dots, s_{i_k})$ is a Prime Implicant path in $T(Pre)$ from the precedent k-depth-first navigation, then from the algorithm, at any transition $(s_{i_j}, s_{i_{j+1}})$, it corresponds (several) state(s) Implication graphs IG_{i_j} for s_{i_j} , and $IG_{i_{j+1}}$ for $s_{i_{j+1}}$ corresponding at any (re)visit of the states.
- There exists a k-depth-first navigation of full $T(Pre)$ following the Prime implicant depth-first-search of the f -tableau, such that if $(s'_{i_0} = Pre, s'_{i_1}, \dots, s'_{i_k})$ is a path of states in $T(Pre)$, then a corresponding Prime implicant path $(s_{i_0} = Pre, s_{i_1}, \dots, s_{i_k})$ is one from k-depth-first navigation of the Prime implicant f -tableau.
- Let $Cl_0(Pre)$, $UC_k(f) \setminus Presence_k(f)$, $AUX_k(f)$, $Learn_k(f, C)$ be the timestamped variables and corresponding clauses. Let $Next_k = \{x_j(X(f)) \Rightarrow x_{j+1}(f) | 0 \leq j < k\}$ be the clauses encoding the state to next prestate derivations. Then there exists a DLL-exploration E of the propositional problem $Cl_0(Pre) \cup UC_k(f) \setminus Presence_k(f) \cup AUX_k(f) \cup Learn_k(f, C) \cup Next_k$ following the k-depth-first navigation of the full $T(Pre)$ but disregarding conflicts which do not occur in the DFS of the SCC in the f -tableau.
- Let E' be the modified exploration of E but by pruning any part of the exploration which contradict any timestamped limit conflict clause.
- Let $E_{Promise}$ be the modified exploration of E' for the boolean SAT problem $Cl_0(Pre)$, $UC_k(f) \setminus Presence_k(f)$, $AUX_k(f)$, $Learn_k(f, C)$, $Next_k, x_k(Promise)$ without learning. Furthermore it non chronologically backtracks. It also considers only conflicts of the form $\{x_k(op(Promise)) ; \neg x_k(op(Promise))\}$. Then clearly $E_{Promise}$ does not find any solution because the promise is not fulfilled and particularly at step k , ie. the boolean problem is unsatisfiable.

It is now feasible to show that :

- (a) The last conflict C_{last} of $E_{Promise}$ is at level 0. This means that ancestor literals in $\mathcal{A}_k(C_{last})$ with no parent gets a level 0, ie. they correspond to clauses $Core_k$ of length one in $Cl_0(Pre)$, $UC_k(f) \setminus Presence_k(f)$, $AUX_k(f)$, $Learn_k(f, C)$, $Next_k, x_k(op(Promise))$ since there is no learning in $E_{Promise}$. Furthermore $x_k((op(Promise))) \in Core_k$. Finally, $Core_k, UC_k(f) \setminus Presence_k(f), AUX_k(f), Learn_k(f, C), Next_k, x_k(op(Promise))$ is an unsatisfiable core.
- (b) Let $Cf'_k = Core_k \setminus (\{x_k(op(Promise))\} \cup learn_k(f, C))$, then $Cf'_k \subset Cl_0(Pre)$. Let Cf_k be the non timestamped literals. Then if $S \models Cf_k$ and since S is a state of a fair path, then if $p_{S,k}$ is the suffix path from S but truncated of length k , $p_{S,k} \models Core_k \setminus x_k(op(Promise))$, $UC_k(f) \setminus Presence_k(f)$, $AUX_k(f)$, $Learn_k(f, C)$, $Next_k \models \neg x_k(op(Promise))$
- (c) $Cf_k = \{e \in Pre | N(e_0 = e) \rightarrow N(e_1) \rightarrow \dots \rightarrow N(e_k = \neg x_{op(Promise)})\}$, with $N(e_i) \rightarrow N(e_{i+1}) \in T_{black}$ and $N(\neg x_{op(Promise)}) \in SCC\}$

It is then straightforward that if $S \models x_{Promise} \wedge_{k \in \mathbb{N}} Cf_k$ then there is a contradiction since p_S will never realize the operand promise $x_{op(Promise)}$. Furthermore, $\wedge_{k \in \mathbb{N}} Cf_k$ is computed as the set of Pre contained in the backward fixpoint over T_{Black} computing ancestors of any $\neg x_{op(Promise)}$ for all states of the SCC.

Theorem 5 The learned clauses and Limit conflict clauses¹⁷ are fair valid.

sketch of the proof: By chronological induction on the learned clause and limit conflict clause per conflict. First, assume that conflict C is the first, thus the $Learn(f, C) = \emptyset$ at lemma 2. Thus $\neg Cf$ and

¹⁷in case of propositional conflict

$\neg Limit_C$ is fair valid. Assume now that $Learn(f, C)$ are valid. Thanks to lemma 2, it follows that $\neg Cf$ and $\neg Limit(C)$ are fair valid.

Theorem 6 The algorithm terminates, is correct and complete

(**sketch of the proof**): *As long as a state is not known to be bad or in a Bad SCC, then it is recorded¹⁸ to avoid infinite loop. As soon as it is sure that it is a bad state or in a bad SCC, then a clause which will never be forgotten and standing for the bad state is learned. Thus, our algorithm is similar to a depth-first-search of SCC in a LTL tableau [19]. However, as soon as there is a conflict, the algorithm prunes part of the tableau which is sure to lead to a failing state/SCC by, sound learning and backtracking using implication dependencies of conflict.*

Theorem 7 (Extraction of coarse small unsatisfiable core)

If $f = \bigwedge_i f_i$ then $\bigwedge_i \{f_i | x_{f_i} \in \mathcal{A}(C_{last})\}$ is a coarse small unsatisfiable core.

(**sketch of the proof**): If the algorithm terminates with ‘unsat’, the last conflict C_{last} is at level 0. This means that ancestor nodes in $\mathcal{A}(C_{last})$ with no parent gets a level 0, ie. they correspond to some clause in $presence(f) : x_{f_i}$ where $f = \bigwedge_i f_i$ or eventually to some learned clause of the form $\neg x_h$. But since $ICl(f, C_{last}) \setminus Presence(f)$ are fair valid, then $\bigwedge_i \{f_i | x_{f_i} \in \mathcal{A}(C_{last})\}$ is a coarse small unsatisfiable core.

6 Conclusion

In order to detect which compliance rules are conflicting, we have provided a conflict-driven Tableau depth-first-search for LTL. We have shown how it can be used to extract a small unsatisfiable core. Our method is theoretically *EXPTIME* and *EXSPACE*, but although deciding a MU is in $P - SPACE$ no $P - SPACE$ method have been proposed to extract cores yet. Our method does not suffer from cumbersome timestamped variables, handling of incrementation, searching upper bound for UMC. Implementation is ongoing work. Three enhancements of the method would be to study a QBF-encoding of our method and analyzes if the learning we propose is easy for QBF solvers to learn. Other ways could be to use symbolic DFS [3] or alternating Büchi automata. Detecting conflicts in rules is critical for human interactive contract management systems. Moreover, our method pinpoints temporal issues in any automatic tool which is sensitive to the consistency of many evolving heterogeneous policies such as regulatory laws, internal business rules, security or privacy. The extension of our method to deontic modality [4, 10] used in contracts appears straightforward, and we are also focusing on this issue.

References

- [1] Rajeev Alur & Thomas A. Henzinger (1991): *Logics and Models of Real Time: A Survey*. In: *REX Workshop*, pp. 74–106, doi:10.1007/BFb0031988.
- [2] Armin Biere, Alessandro Cimatti, Edmund M. Clarke & Yunshan Zhu (1999): *Symbolic Model Checking without BDDs*. In: *TACAS*, pp. 193–207, doi:10.1007/3-540-49059-0_14.
- [3] Armin Biere, Edmund M. Clarke & Yunshan Zhu: *Multiple State and Single State Tableaux for Combining Local and Global Model Checking*. doi:10.1007/3-540-48092-7_8.
- [4] Jan Broersen, Frank Dignum, Virginia Dignum & John-Jules Ch. Meyer (2004): *Designing a Deontic Logic of Deadlines*. In: *DEON*, doi:10.1007/978-3-540-25927-5_5.
- [5] Jerry R. Burch & all. (1992): *Symbolic Model Checking: 10²⁰ States and Beyond*. *Inf. Comput.* 98(2), pp. 142–170, doi:10.1016/0890-5401(92)90017-A.

¹⁸by a hash table for instance

- [6] A. Cimatti, M. Roveri, V. Schuppan & S. Tonetta (2007): *Boolean Abstraction for Temporal Logic Satisfiability*. In: CAV, pp. 532–546, doi:10.1007/978-3-540-73368-3_53.
- [7] Elio Damaggio, Alin Deutsch & Victor Vianu (2011): *Artifact systems with data dependencies and arithmetic*. In: ICDT, pp. 66–77, doi:10.1145/1938551.1938563.
- [8] Martin Davis, George Logemann & Donald W. Loveland (1962): *A machine program for theorem-proving*. *Commun. ACM* 5(7), pp. 394–397, doi:10.1145/368273.368557.
- [9] E. Emerson (1990): *Temporal and Modal Logic*. *HTCS, Volume B: Formal Models and Semantics (B)*.
- [10] Stephen Fenech, Gordon J. Pace & Gerardo Schneider (2009): *Automatic Conflict Detection on Contracts*. In: ICTAC, pp. 200–214, doi:10.1007/978-3-642-03466-4_13.
- [11] Michael J. Fischer & Richard E. Ladner (1979): *Propositional Dynamic Logic of Regular Programs*. *J. Comput. Syst. Sci.* 18(2), pp. 194–211, doi:10.1016/0022-0000(79)90046-1.
- [12] Michael Fisher (1991): *A Resolution Method for Temporal Logic*. In: IJCAI, pp. 99–104.
- [13] Malay K. Ganai, Aarti Gupta & Pranav Ashar (2005): *Beyond safety: customized SAT-based model checking*. In: DAC, pp. 738–743, doi:10.1145/1065579.1065773.
- [14] Rob Gerth, Doron Peled, Moshe Y. Vardi & Pierre Wolper (1995): *Simple on-the-fly automatic verification of linear temporal logic*. In: PSTV, pp. 3–18.
- [15] Aditya Ghose & George Koliadis (2007): *Auditing Business Process Compliance*. In: ICSOC, pp. 169–180, doi:10.1007/978-3-540-74974-5_14.
- [16] C. Giblin, A. Liu, S. Müller & B. Pfitzmann (2005): *Regulations Expressed As Logical Models (REALM)*. In: JURIX, pp. 37–48.
- [17] Keijo Heljanko, Tommi A. Junttila & Timo Latvala (2005): *Incremental and Complete Bounded Model Checking for Full PLTL*. In: CAV, pp. 98–111, doi:10.1007/11513988_10.
- [18] Toni Jussila & Armin Biere (2007): *Compressing BMC Encodings with QBF*. *Electr. Notes Theor. Comput. Sci.* 174(3), pp. 45–56, doi:10.1016/j.entcs.2006.12.022.
- [19] Y. Kesten, Z. Manna, H. McGuire & A. Pnueli (1993): *A Decision Algorithm for Full Propositional Temporal Logic*. In: CAV, pp. 97–109, doi:10.1007/3-540-56922-7_9.
- [20] Annapaola Marconi & Marco Pistore (2009): *Synthesis and Composition of Web Services*. In: SFM, pp. 89–157, doi:10.1007/978-3-642-01918-0_3.
- [21] M. Montali, M. Pesic, W. v.d. Aalst, F. Chesani & P. Mello (2010): *Declarative specification and verification of service choreographies*. *TWEB* 4(1), doi:10.1145/1658373.1658376.
- [22] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang & S. Malik (2001): *Chaff: Engineering an Efficient SAT Solver*. In: DAC, pp. 530–535. Available at <http://www.citeulike.org/user/pwais/article/3403622>.
- [23] Viktor Schuppan (2010): *Towards a notion of unsatisfiable and unrealizable cores for LTL*. *Science of Computer Programming* doi:10.1016/j.scico.2010.11.004.
- [24] Mary Sheeran, Satnam Singh & Gunnar Stålmarck (2000): *Checking Safety Properties Using Induction and a SAT-Solver*. In: FMCAD, pp. 108–125, doi:10.1007/3-540-40922-X_8.
- [25] Robert Endre Tarjan (1972): *Depth-First Search and Linear Graph Algorithms*. *SIAM J. Comput.* 1(2), pp. 146–160, doi:10.1137/0201010.
- [26] K. Xu, Y. Liu & C. Wu (2008): *BPSL Modeler - Visual Notation Language for Intuitive Business Property Reasoning*. *ENTCS* 211, doi:10.1016/j.entcs.2008.04.043.
- [27] L. Zhang, C. Madigan, M. Moskewicz & S. Malik (2001): *Efficient Conflict Driven Learning in Boolean Satisfiability Solver*. In: ICCAD, pp. 279–285. Available at <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.2715>.
- [28] L. Zhang & S. Malik (2003): *Extracting small unsatisfiable cores from unsatisfiable Boolean formula*. In: *In Prelim. Proc. Sixth Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'03)*.